Pascal Pons Follow

SOLVING CONNECT FOUR

Part 6 – Bitboard

We wil see in this part how to use a bitmap encoding of positions to reduce significantly the computation time.

Binary representation of a position

Board positions can be stored in a compact an efficient way using W*(H+1) bits for a board of height H and width W.

Each column is encoded by H bits corresponding to its cells plus an extra bit on top of the column. A 7x6 bord will use 49 bits with the following bit order:

```
.  .  .  .  .  .  .
5 12 19 26 33 40 47
4 11 18 25 32 39 46
3 10 17 24 31 38 45
2  9 16 23 30 37 44
1  8 15 22 29 36 43
0  7 14 21 28 35 42
```

Current player’s stone are encoded as 1, opponent’s stones are encoded as 0. To identify the empty cells, an extra 1 is added on the lowest empty cell of each each column, all other bits above it are set to 0. Note that we need the extra bit per column to encode full columns. This encoding is unambiguous and will be used later as an unique key to represent a position.

Here is an example of encoding (x is current player):

```
          0000000
.......   0001000
...o...   0010000
..xx...   0011000
..ox...   0001100
..oox..   0000110
..oxxo.   1101101
```

However, in order to compute efficiently operations, positions will be stored using two separated bitmaps, one containing only the stones of the current player and one mask identifying the non-empty cells. We can get the non ambiguous key defined above using the following formula:

```
key = position + mask + bottom

board     position  mask      bottom    key
          0000000   0000000   0000000   0000000
.......   0000000   0000000   0000000   0001000
...o...   0000000   0001000   0000000   0010000
..xx...   0011000   0011000   0000000   0011000
..ox...   0001000   0011000   0000000   0001100
..oox..   0000100   0011100   0000000   0000110
..oxxo.   0001100   0011110   1111111   1101101
```

Playing a move

To check if we can play a column we just check if the last cell of the colum is free using the non-empty cell mask:

```
bool canPlay(int col) const
{
  return (mask & top_mask(col)) == 0;
}

static uint64_t top_mask(int col) {
  return (UINT64_C(1) << (HEIGHT - 1)) << col*(HEIGHT+1);
}
```

To play a column, we first switch the bits of current and opponent players by XORing current_position with mask. Then we add the extra bit to mask corresponding to the played cell:

```
void play(int col)
{
  current_position ^= mask;
  mask |= mask + bottom_mask(col);
  moves++;
}

static uint64_t bottom_mask(int col) {
  return UINT64_C(1) << col*(HEIGHT+1);
}
```

Checking for aligment

Checking for alignment using a bitmap of the stones of a player can be perform using the following bitwise operations to check successively the 4 possible directions:

```
static bool alignment(uint64_t pos) {
  // horizontal
  uint64_t m = pos & (pos >> (HEIGHT+1));
  if(m & (m >> (2*(HEIGHT+1)))) return true;

  // diagonal 1
  m = pos & (pos >> HEIGHT);
  if(m & (m >> (2*HEIGHT))) return true;

  // diagonal 2
  m = pos & (pos >> (HEIGHT+2));
  if(m & (m >> (2*(HEIGHT+2)))) return true;

  // vertical;
  m = pos & (pos >> 1);
  if(m & (m >> 2)) return true;

  return false;
}
```

Here is the implementation of the new Position class, and the full source code corresponding to this part.

Benchmark

Bitmap encoding of positions only improves execution time (about 6-fold) while exploring exactly the same nodes.

Solver	Test Set name	mean time	mean nb pos	K pos/s
Bitboard (strong solver)	End-Easy	8.55 μs	139.7	16,334
Bitboard (strong solver)	Middle-Easy	33.31 ms	2,081,790	62,504
Bitboard (strong solver)	Middle-Medium	644 ms	40,396,700	62,727
Bitboard (strong solver)	Begin-Easy	N/A	N/A	N/A
Bitboard (strong solver)	Begin-Medium	N/A	N/A	N/A
Bitboard (strong solver)	Begin-Hard	N/A	N/A	N/A
Bitboard (weak solver)	End-Easy	6.708 μs	107.1	15,973
Bitboard (weak solver)	Middle-Easy	14.69 ms	927,943	63,149
Bitboard (weak solver)	Middle-Medium	370.3 ms	23,685,400	63,968
Bitboard (weak solver)	Begin-Easy	N/A	N/A	N/A
Bitboard (weak solver)	Begin-Medium	N/A	N/A	N/A
Bitboard (weak solver)	Begin-Hard	N/A	N/A	N/A

Tutorial plan

SOLVING CONNECT FOUR

Updated: March 04, 2017

Previous **Next**